

[illegible][illegible]

```
XX      XX      AAAAAA  DDDDDDDD  RRRRRRRR  IIIIII  VV      VV  EEEEEEEEE  RRRRRRRR
XX      XX      AAAAAA  DDDDDDDD  RRRRRRRR  IIIIII  VV      VV  EEEEEEEEE  RRRRRRRR
XX      XX      AA      AA  DD      DD  RR      RR  RR      RR  EE      EE  RR      RR
XX      XX      AA      AA  DD      DD  RR      RR  RR      RR  EE      EE  RR      RR
  XX  XX      AA      AA  DD      DD  RR      RR  RR      RR  EE      EE  RR      RR
  XX  XX      AA      AA  DD      DD  RR      RR  RR      RR  EE      EE  RR      RR
    XX  XX      AA      AA  DD      DD  RRRRRRRR  RR      RR  EE      EE  RRRRRRRR
    XX  XX      AA      AA  DD      DD  RRRRRRRR  RR      RR  EE      EE  RRRRRRRR
  XX  XX      AAAAAAAAAA  DD      DD  RR      RR  RR      RR  EEEEEEEEE  RR      RR
  XX  XX      AAAAAAAAAA  DD      DD  RR      RR  RR      RR  EEEEEEEEE  RR      RR
XX      XX      AA      AA  DD      DD  RR      RR  RR      RR  EE      EE  RR      RR
XX      XX      AA      AA  DD      DD  RR      RR  RR      RR  EE      EE  RR      RR
XX      XX      AA      AA  DDDDDDDD  RR      RR  RR      RR  EEEEEEEEE  RR      RR
XX      XX      AA      AA  DDDDDDDD  RR      RR  RR      RR  EEEEEEEEE  RR      RR
```

```
MM      MM      AAAAAA  RRRRRRRR
MM      MM      AAAAAA  RRRRRRRR
MMMM  MMMM  AA      AA  RR      RR
MMMM  MMMM  AA      AA  RR      RR
MM  MM  MM  AA      AA  RR      RR
MM  MM  MM  AA      AA  RR      RR
MM      MM  AA      AA  RRRRRRRR
MM      MM  AA      AA  RRRRRRRR
MM      MM  AAAAAAAAAA  RR      RR
MM      MM  AAAAAAAAAA  RR      RR
MM      MM  AA      AA  RR      RR
MM      MM  AA      AA  RR      RR
MM      MM  AA      AA  RR      RR
MM      MM  AA      AA  RR      RR
```

.TITLE XADRIVER - VAX/VMS DR11-W DRIVER  
.IDENT 'V04-001'

COPYRIGHT (c) 1978, 1980, 1982, 1984 BY  
DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.  
ALL RIGHTS RESERVED.

THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  
ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE  
INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER  
COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  
OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY  
TRANSFERRED.

THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE  
AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT  
CORPORATION.

DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS  
SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.

FACILITY:

VAX/VMS Executive, I/O Drivers

ABSTRACT:

This module contains the DR11-W driver:

Tables for loading and dispatching  
Controller initialization routine  
FDT routine  
The start I/O routine  
The interrupt service routine  
Device specific Cancel I/O  
Error logging register dump routine

ENVIRONMENT:

Kernal Mode, Non-paged

AUTHOR:

C. A. Sameulson 10-JAN-79

MODIFIED BY:

V04-001 JLV0395 Jake VanNoy 6-SEP-1984  
Add AVL bit to DEVCHAR.

V03-006 TMK0001 Todd M. Katz 07-Dec-1983  
Fix a broken branch.

V03-005 JLV0304 Jake VanNoy 24-AUG-1983  
Several bug fixes. All word writes to XA\_CSR now have  
ATTN set so as to prevent lost interrupts. Attention  
AST list is synchronized at device IPL in DEL ATTNAST.  
Correct status is returned on a set mode ast that  
is returns through EXESFINISHIO. REQCOM's are always  
done at FIPL. Signed division that prevented full size  
transfers has been fixed.

V03-004 KDM0059 Kathleen D. Morse 14-Jul-1983  
Change time-wait loops to use new TIMEDWAIT macro.  
Add \$DEVDEF.

V03-003 KDM0002 Kathleen D. Morse 28-Jun-1982  
Added \$DYNDEF, \$DCDEF, and \$SSDEF.

--



## .SBTTL External and local symbol definitions

## ; External symbols

\$ACBDEF	; AST control block
\$CRBDEF	; Channel request block
\$DCDEF	; Device types
\$DDBDEF	; Device data block
\$DEVDEF	; Device characteristics
\$DPTDEF	; Driver prolog table
\$DYNDEF	; Dynamic data structure types
\$EMBDEF	; EMB offsets
\$IDBDEF	; Interrupt data block
\$IODEF	; I/O function codes
\$IPLDEF	; Hardware IPL definitions
\$IRPDEF	; I/O request packet
\$PRDEF	; Internal processor registers
\$PRIDDEF	; Scheduler priority increments
\$SSDEF	; System status codes
\$UCBDEF	; Unit control block
\$VECDDEF	; Interrupt vector block
\$XADEF	; Define device specific characteristics

## ; Local symbols

## ; Argument list (AP) offsets for device-dependent QIO parameters

P1	= 0	; First QIO parameter
P2	= 4	; Second QIO parameter
P3	= 8	; Third QIO parameter
P4	= 12	; Fourth QIO parameter
P5	= 16	; Fifth QIO parameter
P6	= 20	; Sixth QIO parameter

## ; Other constants

XA_DEF_TIMEOUT	= 10	; 10 second default device timeout
XA_DEF_BUFSIZ	= 65535	; Default buffer size
XA_RESET_DELAY	= <<2+9>/10>	; Delay N microseconds after RESET ; (rounded up to 10 microsec intervals)

## ; DR11-W definitions that follow the standard UCB fields

; \*\*\* N O T E \*\*\* ORDER OF THESE UCB FIELDS IS ASSUMED

\$DEFINI	UCB	
\$DEF	UCBSL_XA_ATTEN	; Attention AST listhead
\$DEF	UCBSW_XA_CSRTMP	; Temporary storage of CSR image
\$DEF	UCBSW_XA_BARTMP	; Temporary storage of BAR image
\$DEF	UCBSW_XA_CSR	; Saved CSR on interrupt
\$DEF	UCBSW_XA_EIR	; Saved EIR on interrupt

```

$DEF      UCBSW_XA_IDR      .BLKW 1      ; Saved IDR on interrupt
$DEF      UCBSW_XA_BAR      .BLKW 1      ; Saved BAR register on interrupt
$DEF      UCBSW_XA_WCR      .BLKW 1      ; Saved WCR register on interrupt
$DEF      UCBSW_XA_ERROR    .BLKW 1      ; Saved device status flag
$DEF      UCBSL_XA_DPR      .BLKW 1      ; Data Path Register contents
$DEF      UCBSL_XA_FMPR     .BLKW 1      ; Final Map Register contents
$DEF      UCBSL_XA_PMPR     .BLKW 1      ; Previous Map Register contents
$DEF      UCBSW_XA_DPRN     .BLKW 1      ; Saved Datapath Register Number
                                          ; And Datapath Parity error flag

; Bit positions for device-dependent status field in UCB
$VFIELD   UCB,0,<-
          <ATTNAST,,M>,-      ; UCB device specific bit definitions
          <UNEXPT,,M>,-      ; ATTN AST requested
          >                  ; Unexpected interrupt received

UCBSK_SIZE=
$DEFEND   UCB

; Device register offsets from CSR address

$DEF      $DEFINI XA          ; Start of DR11-W definitions
$DEF      XA_WCR              ; Word count
$DEF      XA_BAR              .BLKW 1   ; Buffer address
$DEF      XA_CSR              .BLKW 1   ; Control/status

; Bit positions for device control/status register
$EQU      XASK,,0,1,<-      ; Define CSR FNCT bit values
          <FNCT1,2>-
          <FNCT2,4>-
          <FNCT3,8>-
          <STATUSA,2048>-    ; Define CSR STATUS bit values
          <STATUSB,1024>-
          <STATUSC,512>-
          >

$VFIELD   XA_CSR,0,<-
          <GD,,M>,-          ; Control/status register
          <FNCT,3,M>,-       ; Start device
          <XBA,2,M>,-        ; CSR FNCT bits
          <IE,,M>,-          ; Extended address bits
          <RDY,,M>,-         ; Enable interrupts
          <CYCLE,,M>,-       ; Device ready for command
          <STATUS,3,M>,-     ; Starts slave transmit
          >                  ; CSR STATUS bits

```

```

      <MAINT,,M>,-      ; Maintenance bit
      <ATTN,,M>,-      ; Status from other processor
      <NEX,,M>,-       ; Nonexistent memory flag
      <ERROR,,M>,-    ; Error or external interrupt
SDEF   > KA_EIR        ; Error information register
; Bit positions for error information register
$VIELD KA_EIR,0,<-      ; Error information register
      <REGFLG,,M>,-    ; Flags whether EIR or CSR is accessed
      <SPARE,,M>,-     ; Unused - spare
      <BURST,,M>,-     ; Burst mode transfer occurred
      <DLT,,M>,-       ; Time-out for successive burst xfer
      <PAR,,M>,-       ; Parity error during DAT1/P
      <ACLO,,M>,-      ; Power fail on this processor
      <MULTI,,M>,-     ; Multi-cycle request error
      <ATTN,,M>,-     ; ATTN - same as in CSR
      <NEX,,M>,-       ; NEX - same as in CSR
      <ERROR,,M>,-    ; ERROR - same as in CSR
      >
      .BLKW 1
SDEF   KA_IDR          ; Input Data Buffer register
SDEF   KA_ODR          ; Output Data Buffer register
      .BLKW 1
$DEFEND KA             ; End of DR11-W definitions

```



## .SBTTL Device Driver Tables

## ; Driver prologue table

```

DPTAB      -      ; DPT-creation macro
            END=XA END,-      ; End of driver label
            ADAPTER=UBA,-      ; Adapter type
            FLAGS=DPTSM_SVP,-      ; Allocate system page table
            UCBSIZE=UCBSK_SIZE,-      ; UCB size
            NAME=XADriver      ; Driver name
DPT_STORE INIT      ; Start of load
            ; initialization table
DPT_STORE UCB,UCBSB_FIPL,B,8      ; Device fork IPL
DPT_STORE UCB,UCBSB_DIPL,B,22      ; Device interrupt IPL
DPT_STORE UCB,UCBSL_DEVCHAR,L,<-      ; Device characteristics
            DEVSM_AVL!,-      ; Available
            DEVSM_RTM!,-      ; Real Time device
            DEVSM_ELGL,-      ; Error Logging enabled
            DEVSM_IDV!,-      ; input device
            DEVSM_ODV>      ; output device
DPT_STORE UCB,UCBSB_DEVCLASS,B,DC$ REALTIME      ; Device class
DPT_STORE UCB,UCBSB_DEVTYPE,B,DT$_DR11W      ; Device Type
DPT_STORE UCB,UCBSB_DEVBUFSIZ,W,-      ; Default buffer size
            XA DEF BUFSIZ
DPT_STORE REINIT      ; Start of reload
            ; initialization table
DPT_STORE DDB,DDBSL_DDT,D,XASDDT      ; Address of DDT
DPT_STORE CRB,CRBSL_INTD+4,D,-      ; Address of interrupt
            XA INTERRUPT      ; service routine
DPT_STORE CRB,CRBSL_INTD+VECSL_INITIAL,-      ; Address of controller
            D,XA CONTROL_INIT      ; initialization routine
DPT_STORE END      ; End of initialization
            ; tables

```

## ; Driver dispatch table

```

DDTAB      -      ; DDT-creation macro
            DEVNAM=XA,-      ; Name of device
            START=XA START,-      ; Start I/O routine
            FUNCTB=XA_FUNCTABLE,-      ; FDT address
            CANCEL=XA_CANCEL,-      ; Cancel I/O routine
            REGDMP=XA_REGDUMP,-      ; Register dump routine
            DIAGBF=<<13*4>+<<3+5+1>*4>>,-      ; Diagnostic buffer size
            ERLGBF=<<13*4>+<1*4>+<EMBSL_DV_REGSABV>> ; Error log buffer size

```

## ; Function dispatch table

```

XA_FUNCTABLE:      ; FDT for driver
FUNCTAB      -      ; Valid I/O functions
            <READPBLK,READLBLK,READVBLK,WRITEPBLK,WRITELBLK,WRITEVBLK,-
            SETMODE,SETCHAR,SENSEMODE,SENSECHAR>
FUNCTAB      ; No buffered functions
FUNCTAB XA_READ_WRITE,-      ; Device-specific FDT
            <READPBLK,READLBLK,READVBLK,WRITEPBLK,WRITELBLK,WRITEVBLK>
FUNCTAB +EXESREAD,<READPBLK,READLBLK,READVBLK>

```



FUNCTAB \*EX\$WRITE,<WRITEPBLK,WRITEBLK,WRITEVBLK>  
FUNCTAB XA SETMODE,<SETMODE,SETCHAR>  
FUNCTAB \*EX\$SENSEMODE,<SENSEMODE,SENSECHAR>

## .SBTTL KA\_CONTROL\_INIT, Controller initialization

++  
 KA\_CONTROL\_INIT, Called when driver is loaded, system is booted, or power failure recovery.

## Functional Description:

- 1) Allocates the direct data path permanently
- 2) Assigns the controller data channel permanently
- 3) Clears the Control and Status Register
- 4) If power recovery, requests device time-out

## Inputs:

R4 = address of CSR  
 R5 = address of IDB  
 R6 = address of DDB  
 R8 = address of CRB

## Outputs:

VECSV\_PATHLOCK bit set in CRBSL\_INTD+VECSB\_DATAPATH  
 UCB address placed into IDBSL\_OWNER

--  
 .

## KA\_CONTROL\_INIT:

MOVL IDBSL\_UCBLST(R5),R0 ; Address of UCB  
 MOVL R0,IDBSL\_OWNER(R5) ; Make permanent controller owner  
 BISW #UCBSM\_ONLINE,UCBSW\_STS(R0) ; Set device status "on-line"

; If powerfail has occurred and device was active, force device time-out.  
 ; The user can set his own time-out interval for each request. Time-  
 ; out is forced so a very long time-out period will be short circuited.

BBS #UCBSV\_POWER,UCBSW\_STS(R0),10\$ ; Branch if powerfail  
 BISB #VECSM\_PATHLOCK,CRBSL\_INTD+VECSB\_DATAPATH(R8) ; Permanently allocate direct datapath

10\$:

BSBW KA\_DEV\_RESET ; Reset DR11W  
 RSB ; Done

.SBTTL KA\_READ\_WRITE, FDT for device data transfers

KA\_READ\_WRITE, FDT for READBLK, READVBLK, READPBLK, WRITEBLK, WRITEVBLK, WRITEPBLK

#### Functional description:

- 1) Rejects QUEUE I/O's with odd transfer count
- 2) Rejects QUEUE I/O's for BLOCK MODE request to UBA Direct Data PATH on odd byte boundary
- 3) Stores request time-out count specified in P3 into IRP
- 4) Stores FNCT bits specified in P4 into IRP
- 5) Stores word to write into ODR from P5 into IRP
- 6) Checks block mode transfers for memory modify access

#### Inputs:

R3 = Address of IRP  
 R4 = Address of PCB  
 R5 = Address of UCB  
 R6 = Address of CCB  
 R8 = Address of FDT routine  
 AP = Address of P1  
 P1 = Buffer Address  
 P2 = Buffer size in bytes  
 P3 = Request time-out period (conditional on IOSM\_TIMED)  
 P4 = Value for CSR FNCT bits (conditional on IOSM\_SETFNCT)  
 P5 = Value for ODR (conditional on IOSM\_SETFNCT)  
 P6 = Address of Diagnostic Buffer

#### Outputs:

R0 = Error status if odd transfer count  
 IRPSL\_MEDIA = Time-out count for this request  
 IRPSL\_SEGVBN = FNCT bits for DR11-W CSR and ODR image

```

KA_READ_WRITE:
BLBC      P2(AP),108      ; Branch if transfer count even
28:      MOVZWL #SS$BADPARAM,R0      ; Set error status code
56:      JMP      G*EX$ABORTIO      ; Abort request
108:     MOVZWL  IRPSW_FUNC(R3),R1      ; Fetch I/O function code
        MOVL    P3(AP),IRPSL_MEDIA(R3) ; Set request specific time-out count
        BBS     #IOSV_TIMED,R1,158     ; Branch if time-out specified
        MOVL    #XA_DEF_TIMEOUT,IRPSL_MEDIA(R3)
158:     BBC     #IOSV_DIAGNOSTIC,R1,208 ; Else set default timeout value
        EXTZV   #IOSV_FCODE,#IOS$FCODE,R1,R1 ; AND out all function modifiers
        CMPB    #IOS_READPBLK,R1      ; If maintenance function, must be
        ; physical I/O read or write
        BEQL    208
        CMPB    #IOS_WRITEPBLK,R1
        BEQL    208
        MOVZWL  #SS$NOPRIV,R0      ; No privilege for operation
  
```



```

208:   BRB      5$                ; Abort request
      EXTZV   #0,#3,P4(AP),R0    ; Get value for FNCT bits
      ASHL    #XA(CSRSV FNCT,R0,IRPSL SEGVBN(R3)) ; Shift into position for CSR
      MOVW    PS(XP),IRPSL_SEGVBN+2(R3) ; Store ODR value for later

```

```

: If this is a block mode transfer, check buffer for modify access
: whether or not the function is read or write. The DR11-W does
: not decide whether to read or write, the users device does.
: For word mode requests, return to read check or write check.

```

```

: If this is a BLOCK MODE request and the UBA Direct Data Path is
: in use, check the data buffer address for word alignment. If buffer
: is not word aligned, reject the request.

```

```

      BBS     #IOSV_WORD,IRPSW_FUNC(R3),30$
      BBS     #XASV_DATAPATH,UCBSL_DEVDEPEND(R5),25$
      BLBS    P1(AP),2$          ; Branch if word mode transfer
      JMP     G*EXESMODIFY       ; Branch if Buffered Data Path in use
25$:   RSB                    ; ODP, branch on bad alignment
30$:   RSB                    ; Check buffer for modify access
      RSB                    ; Return

```

.SBTTL XA\_SETMODE, Set Mode, Set characteristics FDT

++  
XA\_SETMODE, FDT routine to process SET MODE and SET CHARACTERISTICS

Functional description:

If IOSM\_ATTNASt modifier is set, queue attention AST for device  
If IOSM\_DATAPATH modifier is set, queue packet.  
Else, finish I/O.

Inputs:

R3 = I/O packet address  
R4 = PCB address  
R5 = UCB address  
R6 = CCB address  
R7 = Function code  
AP = QIO Parameter List address

Outputs:

If IOSM\_ATTNASt is specified, queue AST on UCB attention AST list.  
If IOSM\_DATAPATH is specified, queue packet to driver.  
Else, use exec routine to update device characteristics

XA\_SETMODE:

MOVZWL IRPSW\_FUNC(R3),R0 ; Get entire function code  
BBC #IOSV\_ATTNASt,R0,20\$ ; Branch if not an ATTN AST

; Attention AST request

PUSHR #\*M<R4,R7>  
MOVAB UCBSL\_XA\_ATTNASt(R5),R7 ; Address of ATTN AST control block list  
JSB G^COM\$SETATTNASt ; Set up attention AST  
POPR #\*M<R4,R7>  
BLBC R0,50\$ ; Branch if error  
BISW #UCBSM\_ATTNASt,UCBSW\_DEVSTS(R5) ; Flag ATTN AST expected.  
BBC #UCBSV\_UNEXPT,UCBSW\_DEVSTS(R5),10\$ ; Deliver AST if unsolicited interrupt

10\$: BSBW DEL\_ATTNASt  
MOVZBL #SS\$NORMAL,R0 ; Set status  
JMP G^EXE\$FINISHIOC ; Thats all for now (clears R1)

; If modifier IOSM\_DATAPATH is set,  
; queue packet. The data path is changed at driver level to preserve  
; order with other requests.

20\$: BBS S^#IOSV\_DATAPATH,R0,30\$ ; If BDP modifier set, queue packet  
JMP G^EXE\$SETCHAR ; Set device characteristics

; This is a request to change data path usage, queue packet

```
308:  CMPL  #IOS_SETCAR,R7      ; Set characteristics?
      BNEQ 458                  ; No, must have the privilege
      JMP  G^EXES$SETMODE      ; Queue packet to start I/O

; Error, abort IO

458:  MOVZWL #SS$_NOPRIV,R0      ; No priv for operation
508:  CLRL  R1
      JMP  G^EXES$ABORTIO      ; Abort IO on error
```



.SBTTL KA\_START, Start I/O routines

++  
KA\_START - Start a data transfer, set characteristics, enable ATTN AST.

Functional Description:

This routine has two major functions:

- 1) Start an I/O transfer. This transfer can be in either word or block mode. The FNCTN bits in the DR11-W CSR are set. If the transfer count is zero, the STATUS bits in the DR11-W CSR are read and the request completed.
- 2) Set Characteristics. If the function is change data path, the new data path flag is set in the UCB.

Inputs:

R3 = Address of the I/O request packet  
R5 = Address of the UCB

Outputs:

R0 = final status and number of bytes transferred  
R1 = value of CSR STATUS bits and value of input data buffer register  
Device errors are logged  
Diagnostic buffer is filled

--

.ENABL LSB

KA\_START:

: Retrieve the address of the device CSR

```
ASSUME IDBSL_CSR EQ 0
MOVL   UCB$LCRB(R5),R4      ; Address of CRB
MOVL   @CRB$LCINTD+VEC$LCIDB(R4),R4 ; Address of CSR
```

: Fetch the I/O function code

```
MOVZWL IRPSW_FUNC(R3),R1      ; Get entire function code
MOVW   R1,UCB$SW_FUNC(R5)     ; Save FUNC in UCB for Error Logging
EXTZV  #IOSV_FCODE,#IOSS_FCODE,R1,R2 ; Extract function field
```

: Dispatch on function code. If this is SET CHARACTERISTICS, we will  
: select a data path for future use.  
: If this is a transfer function, it will either be processed in word  
: or block mode.

```
CMPB   #IOS_SETCHAR,R2      ; Set characteristics?
BNEQ   JS
```

++

: SET CHARACTERISTICS - Process Set Characteristics QIO function

## INPUTS:

XA\_DATAPATH bit in Device Characteristics specifies which data path to use. If bit is a one, use buffered data path. If zero, use direct datapath.

## OUTPUTS:

CRB is flagged as to which datapath to use.  
DEVDEPEND bits in device characteristics is updated  
XA\_DATAPATH = 1 -> buffered data path in use  
XA\_DATAPATH = 0 -> direct data path in use

```

28:  MOVL    UCBSL_CRB(R5),R0          ; Get CRB address
      MOVQ   IRPSL_MEDIA(R3),UCBSB_DEVCLASS(R5) ; Set device characteristics
      BISB   #VECSM_PATHLOCK,CRBSL_INTD+VECSB_DATAPATH(R0)
      BBC    #XASV_DATAPATH,UCBSL_DEVDEPEND(R5),28 ; Were we right?
      BICB   #VECSM_PATHLOCK,CRBSL_INTD+VECSB_DATAPATH(R0) ; Set buffered datapath
      CLRL   R1                      ; Return Success
      MOVZWL #SSB_NORMAL,R0
      REQCOM

```

; If subfunction modifier for device reset is set, do one here

```

38:   BBC    S^#IOSV_RESET,R1,48      ; Branch if not device reset
      BSBW   XA_DEV_RESET             ; Reset DR11-W

```

; This must be a data transfer function - i.e. READ OR WRITE  
; Check to see if this is a zero length transfer.  
; If so, only set CSR FNCT bits and return STATUS from CSR

```

48:   TSTW    UCBSW_BCNT(R5)           ; Is transfer count zero?
      BNEQ   10$                     ; No, continue with data transfer
      BBC    S^#IOSV_SETFNCT,R1,68    ; Set CSR FNCT specified?
      DSBINT
      MOVW   IRPSL_SEGVBN+2(R3),XA_ODR(R4)
      ; Store word in ODR
      MOVZWL XA_CSR(R4),R0
      BICW   #<XA_CSRSM_FNCT!XA_CSRSM_ERROR>,R0
      BISM   IRPSL_SEGVBN(R3),R0
      BISM   #XA_CSRSM_ATTN,R0        ; Force ATTN on to prevent lost interrupt
      MOVW   R0,XA_CSR(R4)
      BBC    #XASV_LINK,UCBSL_DEVDEPEND(R5),58 ; Link mode?
      BICW3   #XASK_FNCT2,R0,XA_CSR(R4) ; Make FNCT bit 2 a pulse

```

```

58:   ENBINT

```

```

68:   BSBW    XA_REGISTER              ; Fetch DR11-W registers
      BLBS   R0-78                   ; If error, then log it
      JSB    G^ERL$DEVICERR          ; Log a device error
78:   JSB     G^IOC$DIAGBUFILL        ; Fill diagnostic buffer if specified
      MOVL   UCBSW_XA_CSR(R5),R1      ; Return CSR and EIR in R1
      MOVZWL UCBSW_XA_ERROR(R5),R0    ; Return status in R0

```

```

BISB    #XA_CSRSM_IE,XA_CSR(R4) ; Enable device interrupts
REQCOM                      ; Request done

```

```

; Build CSR image in R0 for later use in starting transfers

```

```

10$:

```

```

MOVZWL  UCBSW_BCNT(R5),R0      ; Fetch byte count
DIVL3   #2,R0,UCBSL_XA_DPR(R5) ; Make byte count into word count
; Set up UCBSW_CSRTMP used for loading CSR later

```

```

MOVZWL  XA_CSR(R4),R0
BICW    #<XA_CSRSM_FNCT>,R0
BISW    #XA_CSRSM_IETXA_CSRSM_ATTN,R0 ; Set Interrupt Enable and ATTN
BBC     S^#IOSV_SEIFNCT,R1,20$      ; Set FNCT bits in CSR?
BICW    #<XA_CSRSM_FNCT>,R0          ; Yes, Clear previous FNCT bits
BISB    IRPSC_SEGVBN(R3),R0          ; OR in new value
20$:    BBC     S^#IOSV_DIAGNOSTIC,R1,23$ ; Check for maintenance function
BISW    #XA_CSRSM_MAINT,R0          ; Set maintenance bit in CSR image

```

```

; Is this a word mode or block mode request?

```

```

23$:    MOVW    R0,UCBSW_XA_CSRTMP(R5) ; Save CSR image in UCB
BBC     S^#IOSV_WORD,R1,BLOCK_MODE ; Check if word or block mode
BRW     WORD_MODE ; Branch to handle word mode

```



```

**
: BLOCK MODE -- Process a Block Mode (DMA) transfer request

```

```

: FUNCTIONAL DESCRIPTION:

```

```

    This routine takes the buffer address, buffer size, function code,
    and function modifier fields from the IRP. It calculates the UNIBUS
    address, allocates the UBA map registers, loads the DR11-W device
    registers and starts the request.

```

```

--
: Set up UBA
: Start transfer

```

```

BLOCK_MODE:

```

```

: If IOSM_CYCLE subfunction is specified, set CYCLE bit in CSR image

```

```

    BBC    #IOSM_CYCLE,R1,255      ; Set CYCLE bit in CSR?
    BISW    #XA_CSRSM_CYCLE,UCBSW_XA_CSRIMP(R5) ; If yes, or into CSR image

```

```

: Allocate UBA data path and map registers

```

```

255:

```

```

    REQDPR      ; Request UBA data path
    REQMPR      ; Request UBA map registers
    LOADUBA     ; Load UBA map registers

```

```

: Calculate the UNIBUS transfer address for the DR11-W from the UBA
: map register address and byte offset.

```

```

    MOVZWL    UCBSW_BOFF(R5),R1      ; Byte offset in first page of xfer
    MOVL      UCBSL_CRB(R5),R2       ; Address of CRB
    INSV      (CRBSL_INTD+VECSW_MAPREG(R2),#9,#9,R1) ; Insert page number
    EXTZV     #16,#2,R1,R2          ; Extract bits 17:16 of bus address
    ASHL      #XA_CSRSV_XBA,R2,R2    ; Shift extended memory bits for CSR
    BISW      #XA_CSRSM_GO,R2        ; Set "GO" bit into CSR image
    BISW      R2,UCBSW_XA_CSRIMP(R5) ; Set into CSR image we are building
    BICW3     #<XA_CSRSM_GO:XA_CSRSM_CYCLE>,UCBSW_XA_CSRIMP(R5),R0 ; CSR image less "GO" and "CYCLE"
    BICW3     #XASK_FNCT2,UCBSW_XA_CSRIMP(R5),R2 ; CSR image less FNCT bit 2
    MOVW      R1,UCBSW_XA_BARTMP(R5) ; Save BAR for error logging

```

```

: At this juncture:

```

```

    R0 = CSR image less "GO" and "CYCLE"
    R1 = low 16 bits of transfer bus address
    R2 = CSR image less FNCT bit 2
    UCBSL_XA_DPR(R5) = transfer count in words
    UCBSW_XA_CSRIMP(R5) = CSR image to start transfer with

```

```

: Set DR11-W registers and start transfer

```

```

: Note that read-modify-write cycles are NOT performed to the DR11-W CSR.
: The CSR is always written directly into. This prevents inadvertently setting
: the EIR select flag (writing bit 15) if error happens to become true.

```

```

    DSBINT      ; Disable interrupts (powerfail)

```

```

MNEGW  UCBSL_XA_DPR(R5),XA_WCR(R4)      ; Load negative of transfer count
MOVW   R1,XA_BAR(R4)                    ; Load low 16 bits of bus address
MOVW   R0,XA_CSR(R4)                    ; Load CSR image less 'GO' and 'CYCLE'
BBC    #XASV_LINK,UCBSL_DEVDEPEND(R5),268 ; Link mode?
MOVW   R2,XA_CSR(R4)                    ; Yes, load CSR image less 'FNCT' bit 2
BRB    1268                             ; Only if link mode in dev characteristics

268:   MOVW   UCBSW_XA_CSRTMP(R5),XA_CSR(R4) ; Move all bits to CSR

; Wait for transfer complete interrupt, powerfail, or device time-out

1268:   WFIKPCX XA_TIME_OUT,IRPSL_MEDIA(R3) ; Wait for interrupt

; Device has interrupted, FORK

      IOFORK                             ; FORK to lower IPL

; Handle request completion, release UBA resources, check for errors

MOVZWL #SSS NORMAL, -(SP)                ; Assume success, store code on stack
CLRW   UCBSW_XA_DPRN(R5)                 ; Clear DPR number and DPR error flag
PURDPR                                ; Purge UBA buffered data path
BLBS   R0,278                            ; Branch if no datapath error
MOVZWL #SSS PARITY, (SP)                 ; Flag parity error on device
INCB   UCBSW_XA_DPRN+1(R5)               ; Flag DPR error for log
278:   MOVL   R1,UCBSL_XA_DPR(R5)         ; Save data path register in UCB
EXTZV  #VECSV_DATAPATH,-                 ; Get Datapath register no.
      #VECSL_DATAPATH,-                 ; For Error Log
      (RBSL_INTD+VECSL_DATAPATH(R3),R0)
MOVB   R0,UCBSW_XA_DPRN(R5)             ; Save for later in UCB
EXTZV  #9,#7,UCBSW_XA_BAR(R5),R0        ; Low bits, final map register no.
EXTZV  #4,#2,UCBSW_XA_CSR(R5),R1        ; Hi bits of map register no.
INSV   R1,#7,#2,R0                      ; Entire map register number
CMPW   R0,#496                           ; Is map register number in range?
BGTR   288                               ; No, forget it - compound error
MOVL   (R2)[R0],UCBSL_XA_FMPR(R5)       ; Save map register contents
CLRL   UCBSL_XA_PMPR(R5)                 ; Assume no previous map register
DECL   R0                                ; Was there a previous map register?
CMPV   #VECSV_MAPREG,#VECSL_MAPREG,-    ;
      (RBSL_INTD+VECSL_MAPREG(R3),R0)
BGTR   288                               ; No if gtr
288:   MOVL   (R2)[R0],UCBSL_XA_FMPR(R5) ; Save previous map register contents
      RELMPR                             ; Release UBA resources
      RELDPR

; Check for errors and return status

TSTW   UCBSW_XA_WCR(R5)                 ; All words transferred?
BEQL   308                               ; Yes
MOVZWL #SSS OPINCOMPL, (SP)             ; No, flag operation not complete
308:   BBC    #XA_CSRSV_ERROR,UCBSW_XA_CSR(R5),358 ; Branch on CSR error bit
MOVZWL UCBSW_XA_ERROR(R5),(SP)          ; Flag for controller/drive error status
BSBW   XA_DEV_RESET                     ; Reset DR11-W
358:   BLBS   (SP),408                   ; Any errors after all this?

```

```

408: JSB      G^ERLSDEVICERR      : Yes, log them
      BSBW    DEL ATTNAST        : Deliver outstanding ATTN AST's
      JSB      G^IOCSDIAGBUFILL  : Fill diagnostic buffer
      MOVL     (SP)+,R0          : Get final device status
      MULW3    #2,UCBSW,XA,WCR(R5),R1 : Calculate final transfer count
      ADDW     UCBSW,BCNT(R5),R1
      INSV     R1,#16,#16,R0     : Insert into high byte of IOSB
      MOVL     UCBSW,XA,CSR(R5),R1 : Return CSR and EIR in IOSB
      BISB     #XA_CSRBM_IE,XA,CSR(R4) : Enable interrupts
      REQCOM

```



.DSABL LSB

++  
WORD MODE -- Process word mode (interrupt per word) transfer

FUNCTIONAL DESCRIPTION:

Data is transferred one word at a time with an interrupt for each word. The request is handled separately for a write (from memory to DR11-W and a read (from DR11-W to memory). For a write, data is fetched from memory, loaded into the ODR of the DR11-W and the system waits for an interrupt. For a read, the system waits for a DR11-W interrupt and the IDR is transferred into memory. If the unsolicited interrupt flag is set, the first word is transferred directly into memory without waiting for an interrupt.

--  
.ENABL LSB  
WORD\_MODE:

; Dispatch to separate loops on READ or WRITE

CMPB #IOS\_READPBLK,R2 ; Check for read function  
BEQL 30\$

++  
WORD MODE WRITE -- Write (output) in word mode

FUNCTIONAL DESCRIPTION:

Transfer the requested number of words from user memory to the DR11-W ODR one word at a time, wait for interrupt for each word.

--  
10\$:

BSBW MOVFRUSER ; Get two bytes from user buffer  
DSBINT ; Lock out interrupts  
; Flag interrupt expected  
MOVW R1,XA\_ODR(R4) ; Move data to DR11-W  
MOVW UCBSW-XA CSRTMP(R5),XA CSR(R4) ; Set DR11-W CSR  
BBC #XASV\_LINK,UCBSL\_DEVDEPEND(R5) 15\$ ; Link mode?  
BICW3 #XASK\_FNCT2,UCBSW-XA\_CSRTMP(R5),XA CSR(R4) ; Clear interrupt FNCT bit 2  
; Only if Link mode specified

15\$:

; Wait for interrupt, powerfail, or device time-out

WFIKPCW XA\_TIME\_OUTW,IRPSL\_MEDIA(R3)

; Check for errors, decrement transfer count, and loop til complete

IOFORK ; Fork to lower IPL  
BITW #XA\_EIRSM\_NEX!-  
XA\_EIRSM\_MULT!:-

```

                XA_EIRSM_ACLO!-
                XA_EIRSM_PAR!-
                XA_EIRSM_DLT,UCBSW_XA_EIR(R5) ; Any errors?
208:    BEQL    208      ; No, continue
        BRW    408      ; Yes, abort transfer.
        DECV   UCBSL_XA_DPR(R5) ; All words trnasferred?
        BNEQ   108      ; No, loop until finished.

```

```

; Transfer is done, clear interrupt expected flag and FORK
; All words read or written in WORD MODE. Finish I/O.

```

```

RETURN_STATUS:

```

```

228:    JSB     G*IOCSDIAGBUFILL ; Fill diagnostic buffer if present
        BSBW    DEL_ATTNAST      ; Deliver outstanding ATTN AST's
        MOV2WL  #SSB_NORMAL,R0   ; Complete success status
        MULW3   #2,UCBSL_XA_DPR(R5),R1 ; Calculate actual bytes xfered
        SUBW3   R1,UCBSW_BCNT(R5),R1 ; From requested number of bytes
        INSV    R1,#16,#16,R0    ; And place in high word of R0
        MOVL    UCBSW_XA_CSR(R5),R1 ; Return CSR and EIR status
        BISB    #XA_CSRSM_IE,XA_CSR(R4) ; Enable device interrupts
        REQCOM   ; Finish request in exec

```

```

**
WORD MODE READ -- Read (input) in word mode

```

```

FUNCTIONAL DESCRIPTION:

```

```

Transfer the requested number of words from the DR11-W IDR into
user memory one word at a time, wait for interrupt for each word.
If the unexpected (unsolicited) interrupt bit is set, transfer the
first (last received) word to memory without waiting for an
interrupt.
--

```

```

308:    DSBINT   UCBSB_DIPL(R5) ; Lock out interrupts

```

```

; If an unexpected (unsolicited) interrupt has occurred, assume it
; is for this READ request and return value to user buffer without
; waiting for an interrupt.

```

```

        BBCC    #UCBSV_UNEXPT,-
                UCBSW_DEVSTS(R5),328 ; Branch if no unexpected interrupt
        ENBINT   ; Enable interrupts
        BRB     378 ; continue

```

```

328:    SETIPL   #IPLS_POWER
358:

```

```

; Wait for interrupt, powerfail, or device time-out

```

```

        WFIKPCN XA_TIME_OUTW,IRPSL_MEDIA(R3)

```

```

; Check for errors, decrement transfer count and loop until done

```

```

IOFORK                ; Fork to lower IPL
378:  BITM    #XA_EIRSM_NEX!-
        XA_EIRSM_MULT!-
        XA_EIRSM_ACLO!-
        XA_EIRSM_PAR!-
        XA_EIRSM_DLT,UCBSW_XA_EIR(R5) ; Any errors?
        BNEQ  408                ; Yes, abort transfer.
        BSBW  MOVTOUSER          ; Store two bytes into user buffer
; Send interrupt back to sender. Acknowledge we got last word.
        DSBINT
        MOVW  UCBSW_XA_CSRTMP(R5),XA_CSR(R4)
        BBC   #XASV_LINK,UCBSL_DEVDEPEND(R5),388 ; Link mode?
        BICW3 #XASK_FNCT2,UCBSW_XA_CSRTMP(R5),XA_CSR(R4) ; Yes, clear FNCT 2
388:  DECW    UCBSL_XA_DPR(R5)                ; Decrement transfer count
        BNEQ  358                ; Loop until all words transferred
        ENBINT
        BRW   RETURN_STATUS          ; Finish request in common code
; Error detected in word mode transfer
408:  BSBW    DEL_ATTNAST            ; Deliver ATTN AST's
        BSBW  XA_DEV_RESET          ; Error, reset DR11-W
        JSB   G*TOCS$DIAGBUFILL    ; Fill diagnostic buffer if presetn
        JSB   G*ERL$DEVICERR        ; Log device error
        MOVZWL UCBSW_XA_ERROR(R5),R0 ; Set controller/drive status in R0
        BRW   228
        .DSABL  LSB
;
; MOVFRUSER - Routine to fetch two bytes from user buffer.
;
; INPUTS:
;
;   R5 = UCB address
;
; OUTPUTS:
;
;   R1 = Two bytes of data from users buffer
;   Buffer descriptor in UCB is updated.
;
        .ENABL  LSB
MOVFRUSER:
        MOVAL  -(SP),R1              ; Address of temporary stack loc
        MOVZBL #2,R2                ; Fetch two bytes
        JSB    G*{OC$MOVFRUSER      ; Call exec routine to do the deed
        MOVL   (SP)+,R1              ; Retrieve the bytes
        BRB    208                  ; Update UCB buffer pointers
;
; MOVTOUSER - Routine to store two bytes into users Buffer.
;

```

## : INPUTS:

RS = UCB address

UCBSW\_XA\_IDR(RS) = Location where two bytes are saved

## : OUTPUTS:

Two bytes are stored in user buffer and buffer descriptor in  
UCB is updated.

## : MOVTOUSER:

```

MOVAB    UCBSW_XA_IDR(RS),R1      : Address of internal buffer
MOVZBL   #2,R2
JSB      G^1OC$MOVTOUSER          : Call exec
20$:
ADDW     #2,UCBSW_BOFF(RS)         : Update buffer pointers in UCB
BICW     #^(<^X01FF>,UCBSW_BOFF(RS)) : Add two to buffer descriptor
BNEQ     30$,                        : Modulo the page size
ADDL     #4,UCBSL_SVAPTE(RS)       : If NEQ, no page boundary crossed
30$:
RSB
:
.DSABL   LSB

```



.PAGE  
 .SBTTL DR11-W DEVICE TIME-OUT

++  
 DR11-W device TIME-OUT  
 If a DMA transfer was in progress, release UBA resources.  
 For DMA or WORD mode, deliver ATTN AST's, log a device timeout error,  
 and do a hard reset on the controller.

Clear DR11-W CSR  
 Return error status

Power failure will appear as a device time-out

```
--
      .ENABL  LSB
XA_TIME_OUT:                ; Time-out for DMA transfer

      SETIPL UCBSB_FIPL(R5)   ; Lower to FORK IPL
      PURDPR                ; Purge buffered data path in UBA
      RELMPR                ; Release UBA map registers
      RELDPR                ; Release UBA data path
      BRB 10$                ; continue

XA_TIME_OUTW:                ; Time-out for WORD mode transfer

10$:  SETIPL UCBSB_FIPL(R5)   ; Lower to FORK IPL
      MOVL UCBSL_CRB(R5),R4   ; Fetch address of CSR
      MOVL @CRBSL_INTD+VECSL_IDB(R4),R4
      BSBW XA_REGISTER        ; Read DR11-W registers
      JSB  G*TOCSDIAGBUFILL    ; Fill diagnostic buffer
      JSB  G*ERLSDEVICTMO      ; Log device time out
      BSBW DEL_ATTNAST         ; And deliver the AST's
      BSBW XA_DEV_RESET        ; Reset controller
      MOVZWL #SS$_TIMEOUT,R0   ; Assume error status
      BBC  #UCBSV_CANCEL,-     ; Branch if not cancel
      MOVZWL #SS$_CANCEL,R0    ; Set status

20$:  CLRL R1                  ; Clear ATTN AST flags
      CLRW UCBSW_DEVSTS(R5)    ; Clear unit status flags
      BICW #<UCBSM_TIM!UCBSM_INT!UCBSM_TIMEOUT!UCBSM_CANCEL!UCBSM_POWER>,-
      UCBSW_STS(R5)            ; Complete I/O in exec

      REOCOM
      .DSABL  LSB
      .PAGE
```

.SBTTL XA\_INTERRUPT, Interrupt service routine for DR11-W

++ XA\_INTERRUPT, Handles interrupts generated by DR11-W

Functional description:

This routine is entered whenever an interrupt is generated by the DR11-W. It checks that an interrupt was expected. If not, it sets the unexpected (unsolicited) interrupt flag. All device registers are read and stored into the UCB. If an interrupt was expected, it calls the driver back at its Wait For Interrupt point. Deliver ATTN AST's if unexpected interrupt.

Inputs:

00(SP) = Pointer to address of the device IDB  
 04(SP) = saved R0  
 08(SP) = saved R1  
 12(SP) = saved R2  
 16(SP) = saved R3  
 20(SP) = saved R4  
 24(SP) = saved R5  
 28(SP) = saved PSL  
 32(SP) = saved PC

Outputs:

The driver is called at its Wait For Interrupt point if an interrupt was expected.  
 The current value of the DR11-W CSR's are stored in the UCB.

```
--
XA_INTERRUPT:                ; Interrupt service for DR11-W
    MOVL    @ (SP)+, R4        ; Address of IDB and pop SP
    MOVQ    (R4), R4           ; CSR and UCB address from IDB

; Read the DR11-W device registers (WCR, BAR, CSR, EIR, IDR) and store
; into UCB.
    BSBW    XA_REGISTER        ; Read device registers

; Check to see if device transfer request active or not
; If so, call driver back at Wait for Interrupt point and
; Clear unexpected interrupt flag.
20$:    BBCC    #UCBSV_INT,UCBSW_STS(R5),25$
                                ; If clear, no interrupt expected

; Interrupt expected, clear unexpected interrupt flag and call driver
; back.
    BICW    #UCBSM_UNEXPT,UCBSW_DEVSTS(R5)
                                ; Clear unexpected interrupt flag
    MOVL    UCBSL_FR3(R5), R3   ; Restore drivers R3
    JSB     @UCBSL_FPC(R5)      ; Call driver back
```

BRB 30\$

; Deliver ATTN AST's if no interrupt expected and set unexpected  
; interrupt flag.

25\$:

BISW #UCBSM\_UNEXPT,UCBSW\_DEVSTS(R5) ; Set unexpected interrupt flag  
BSBW DEL\_ATTNAST ; Deliver ATTN AST's  
BISB #XA\_CSRSM\_IE,XA\_CSR(R4) ; Enable device interrupts

; Restore registers and return from interrupt

30\$:

POPR #M<R0,R1,R2,R3,R4,R5> ; Restore registers  
REI ; Return from interrupt

.PAGE  
 .SBTTL XA\_REGISTER - Handle DR11-W CSR transfers

++  
 XA\_REGISTER - Routine to handle DR11-W register transfers

INPUTS:

R4 - DR11-W CSR address  
 R5 - UCB address of unit

OUTPUTS:

CSR, EIR, WCR, BAR, IDR, and status are read and stored into UCB.  
 The DR11-W is placed in its initial state with interrupts enabled.

R0 - .true. if no hard error  
 .false. if hard error (cannot clear ATTN)

If the CSR ERROR bit is set and the associated condition can be cleared, then the error is transient and recoverable. The status returned is SSS\_DRVERR.  
 If the CSR ERROR bit is set and cannot be cleared by clearing the CSR, then this is a hard error and cannot be recovered. The returned status is SSS\_CTRLERR.

R0,R1 - destroyed, all other registers preserved.

XA\_REGISTER:

```

MOVZWL #SS$ NORMAL, R0      ; Assume success
MOVZWL XA_CSR(R4), R1        ; Read CSR
MOVW R1, UCBSW_XA_CSR(R5)    ; Save CSR in UCB
BBC #XA_CSR$V_ERROR, R1, 55$ ; Branch if no error
MOVZWL #SS$ DRVERR, R0       ; Assume "drive" error
55$: BICW #C<XA_CSR$M_FNCT>, R1 ; Clear all uninteresting bits for later
      BISB #<XA_CSR$M_ERROR/256>, XA_CSR+1(R4) ; Set EIR flag
      MOVW XA_EIR(R4), UCBSW_XA_EIR(R5) ; Save EIR in UCB
      MOVW R1, XA_CSR(R4)      ; Clear EIR flag and errors
      MOVW XA_CSR(R4), R1      ; Read CSR back
      BBC #XA_CSR$V_ATTN, R1, 60$ ; If attention still set, hard error
MOVZWL #SS$ CTRLERR, R0      ; Flag hard controller error
60$: MOVW XA_IDR(R4), UCBSW_XA_IDR(R5) ; Save IDR in UCB
      MOVW XA_BAR(R4), UCBSW_XA_BAR(R5)
      MOVW XA_WCR(R4), UCBSW_XA_WCR(R5)
      MOVW R0, UCBSW_XA_ERROR(R5) ; Save status in UCB
      RSB

```

.SBTTL XA\_CANCEL, Cancel I/O routine

XA\_CANCEL, Cancels an I/O operation in progress

Functional description:

Flushes Attention AST queue for the user.  
If transfer in progress, do a device reset to DR11-W and finish the request.  
Clear interrupt expected flag.

Inputs:

R2 = negated value of channel index  
R3 = address of current IRP  
R4 = address of the PCB requesting the cancel  
R5 = address of the device's UCB

Outputs:

XA\_CANCEL: ; Cancel I/O

BBCC #UCBSV ATTNAST, -  
UCBSW\_DEVSTS(R5), 208 ; ATTN AST enabled?

; Finish all ATTN AST's for this process.

PUSHR #\*M<R2,R6,R7>  
MOVL R2,R6 ; Set up channel number  
MOVAB UCBSL\_XA\_ATT(R5),R7 ; Address of listhead  
JSB G\*COMSFLOSHATTNS ; Flush ATTN AST's for process  
POPR #\*M<R2,R6,R7>

; Check to see if a data transfer request is in progress  
; for this process on this channel

208:

DSBINT UCBSB\_DIPL(R5) ; Lock out device interrupts  
JSB G\*IOBCANCELIO ; Check if transfer going  
BBC #UCBSV\_CANCEL, -  
UCBSW\_STS(R5), 308 ; Branch if not for this guy

; Force timeout

CLRL UCBSL\_DUETIM(R5) ; clear timer  
BISW #UCBSM\_TIM,UCBSW\_STS(R5) ; set timed  
BICW #UCBSM\_TIMEOUT, -  
UCBSW\_STS(R5) ; Clear timed out

308:

ENBINT ; Lower to FORK IPL  
RSB ; Return



.PAGE  
 .SBTTL DEL\_ATTNASt, Deliver ATTN AST's

DEL\_ATTNASt, Deliver all outstanding ATTN AST's

Functional description:

This routine is used by the DR11-W driver to deliver all of the outstanding attention AST's. It is copied from COM\$DELATTNASt in the exec. In addition, it places the saved value of the DR11-W CSR and Input Data Buffer Register in the AST paramater.

Inputs:

R5 = UCB of DR11-W unit

Outputs:

R0,R1,R2 Destroyed  
 R3,R4,R5 Preserved

```

DEL_ATTNASt:
DSBINT  UCBSB DIPL(R5)          ; Device IPL
BBCC    #UCBSV_ATTNASt,UCBSW_DEVSTS(R5),30$
                                ; Any ATTN AST's expected?
10$:    PUSHR  #^M<R3,R4,R5>      ; Save R3,R4,R5
        MOVL  8(SP),R1           ; Get address of UCB
        MOVAB UCBSL_XA_ATTNASt(R1),R2 ; Address of ATTN AST listhead
        MOVL  (R2),R5           ; Address of next entry on list
        BEQL  20$              ; No next entry, end of loop
        BICW  #UCBSM_UNEXPT,UCBSW_DEVSTS(R1) ; Clear unexpected interrupt flag
        MOVL  (R5),(R2)         ; Close list
        MOVW  UCBSW_XA_IDR(R1),ACBSL_KAST+6(R5)
                                ; Store IDR in AST paramater
        MOVW  UCBSW_XA_CSR(R1),ACBSL_KAST+4(R5)
                                ; Store CSR in AST paramater
        PUSHAB B*10$           ; Set return address for FORK
        FORK                                ; FORK for this AST

; AST fork procedure
        MOVQ  ACBSL_KAST(R5),ACBSL_AST(R5)
                                ; Re-arrange entries
        MOVB  ACBSL_KAST+8(R5),ACBSB_RMOD(R5)
        MOVL  ACBSL_KAST+12(R5),ACBSL_PID(R5)
        CLRL  ACBSL_KAST(R5)
        MOVZBL #PRI$-IOCOM,R2    ; Set up priority increment
        JMP   G^SCH$QAST         ; Queue the AST

20$:    POPR   #^M<R3,R4,R5>      ; Restore registers
30$:    ENBINT
        RSB                      ; Return
  
```

.PAGE

.SBTTL XA\_REGDUMP - DR11-W register dump routine

```

**
XA_REGDUMP - DR11-W Register dump routine.

```

```

This routine is called to save the controller registers in a specified
buffer. It is called from the device error logging routine and from the
diagnostic buffer fill routine.

```

## Inputs:

```

R0 - Address of register save buffer
R4 - Address of Control and Status Register
R5 - Address of UCB

```

## Outputs:

```

The controller registers are saved in the specified buffer.

```

```

CSRTMP - The last command written to the DR11-W CSR by
          by the driver.
BARTMP - The last value written into the DR11-W BAR by
          the driver during a block mode transfer.
CSR - The CSR image at the last interrupt
EIR - The EIR image at the last interrupt
IDR - The IDR image at the last interrupt
BAR - The BAR image at the last interrupt
WCR - Word count register
ERROR - The system status at request completion
PDRN - UBA Datapath Register number
DPR - The contents of the UBA Data Path register
FMPR - The contents of the last UBA Map register
PMRP - The contents of the previous UBA Map register
DPRF - Flag for purge datapath error
        0 = no purger datapath error
        1 = parity error when datapath was purged

```

```

Note that the values stored are from the last completed transfer
operation. If a zero transfer count is specified, then the
values are from the last operation with a non-zero transfer count.

```

## XA\_REGDUMP:

```

MOVZBL #11,(R0)+ ; Eleven registers are stored.
MOVAB UCB$W_XA_CSRTMP(R5),R1 ; Get address of saved register images
MOVZBL #8,R2 ; Return 8 registers here
10%: MOVZWL (R1)+,(R0)+
      SOGTR R2,10% ; Move them all
      MOVZBL UCB$W_XA_DPRN(R5),(R0)+ ; Save Datapath Register number
      MOVZBL #3,R2 ; And 3 more here
20%: MOVL (R1)+,(R0)+ ; Move UBA register contents
      SOGTR R2,20%
      MOVZBL UCB$W_XA_DPRN+1(R5),(R0)+ ; Save Datapath Parity Error Flag
      RSB

```

.PAGE  
.SBTTL XA\_DEV\_RESET - Device reset DR11-W

++  
XA\_DEV\_RESET - DR11-W Device reset routine

This routine raises IPL to device IPL, performs a device reset to the required controller, and re-enables device interrupts.

Inputs:

R4 - Address of Control and Status Register  
R5 - Address of UCB

Outputs:

Controller is reset, controller interrupts are enabled

--

XA\_DEV\_RESET:

PUSHR #M<R0,R1,R2> ; Save some registers  
DSBINT ; Raise IPL to lock all interrupts  
MOVB #XA\_CSR&M\_MAINT/256,XA\_CSR+1(R4)  
CLRB XA\_CSR+1(R4)

; \*\*\* Must delay here depending on reset interval

TIMEDWAIT TIME=#XA\_RESET\_DELAY ; No. of 10 micro-sec intervals to wait

MOVB #XA\_CSR&M\_IE,XA\_CSR(R4) ; Re-enable device interrupts  
ENBINT ; Restore IPL  
POPR #M<R0,R1,R2> ; Restore registers

RSB

XA\_END: ; End of driver label  
.END



0107 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY